GSI Helmholtzzentrum für Schwerionenforschung GmbH

Gemfony scientific

The Geneva library: Using massively parallel processing to solve complex optimisation problems

Kilian Schwarz / Matthias Lutz
Rüdiger Berlich / Ariel Garcia / Jan Knedlik
Denis Bertini / Jannis Geuppert

GSI Helmholtzzentrum für Schwerionenforschung GmbH
Gemfony scientific UG (haftungsbeschränkt)

25.09.2020 / GSI Panda Workshop

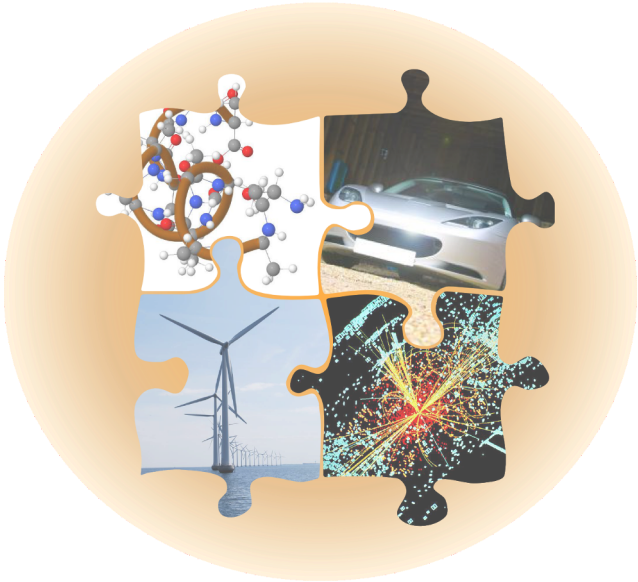
Kontakt:
k.schwarz@gsi.de
r.berlich@gemfony.eu

Table of Contents

- Introduction and functionality
- Using Geneva
- Geneva application in Hadron theory
- GSI contributions and benchmarking
- Lessons Learned
- Conclusion and outlook

The Geneva Library Collection

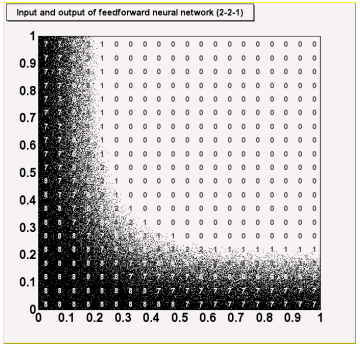
- Generic C++ framework for the search for **optimized solutions** of technical and scientific problems
- Covering **Evolutionary Algorithms**, Swarm Algorithms, Simulated Annealing, Parameter Scans and Gradient Descents
- Data structures allow direct interaction between different optimization algorithms with **just one problem description**
- **Inter-parameter constraints (x+y < 1) possible**
- Support for **many-core systems** as well as parallel and distributed environments
- Available under the Apache License v2 (get it from https://github.com/gemfony/geneva)

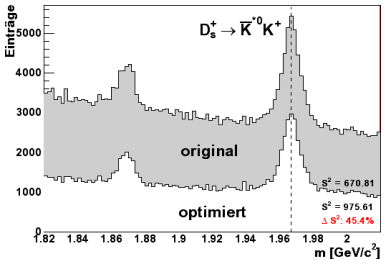


Picture credits: see last page

Some History

- Started as a means of training neural networks 1994 (Ruhr-Universität Bochum, EP1, Crystal Barrel)
- Extended to optimize particle physics analysis (Ruhr-Universität Bochum, EP1, BaBar)
- Rewrite at Karlsruhe Institute of Technology and subject of a Spin-Off
- Usage at GSI and in Industry
- Geneva (and its predecessors) have grown over time, from some 1000 LOC to over 130000 LOC
- **The approach has proven to be highly useful, and we would appreciate your help in further development**

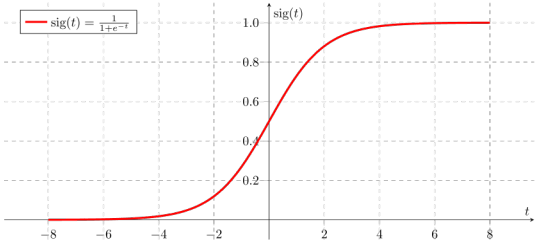




Source: Gemfony

Why here?

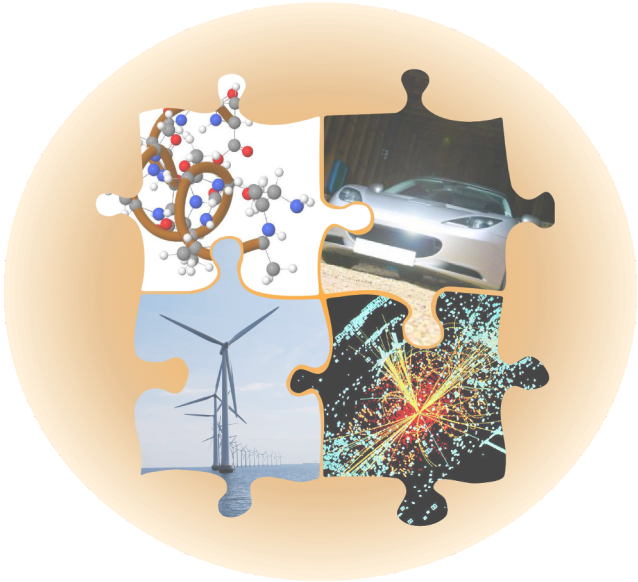
- Standard EA arguably not suitable for large scale ML
  - Lacks „partial training“
  - But has the necessary infrastructure to deal with optimization cycles, stop-criteria, distribution of work loads, ...
  - May cascade algorithms
  - ML might be implementable as a „YAO“ (yet another optimization algorithm)
- Could be extended to perform Neuro-Evolution ?
  - „Started life as a means of training FF neural networks“ (still available as an example)
  - Was used from the very beginning to also train the architecture of feedforward networks
  - Geneva combines floating point, boolean and integer parameters
- All ideas are welcome!



Source: Wikipedia
CC0, MartinThoma

Our Vision

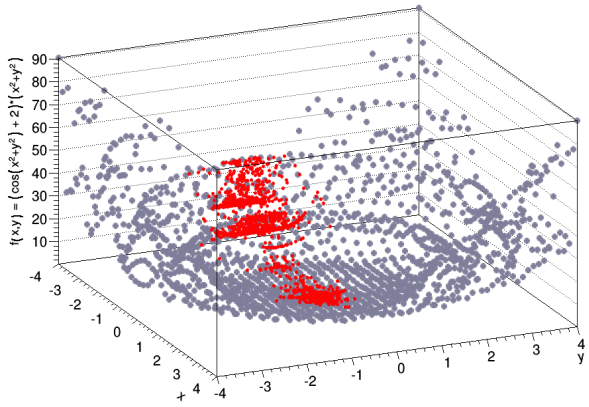
- To create a common Open Source optimization framework, constantly developed and extended by physicists, computer scientists and engineers according to professional standards, covering the most recent algorithms for massively parallel optimization studies in research and industry
- To concentrate, activate and leverage fragmented knowledge in the field of parametric optimization to enable each other solving even the most daunting optimization problems
- To identify new deployment scenarios, e.g. in ML, modelling and simulations, ...



Picture credits: see last page

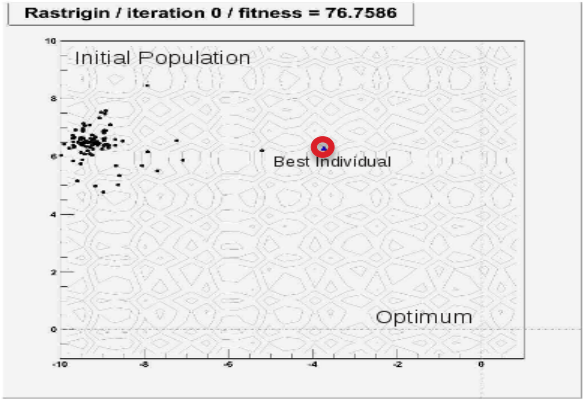
Dealing with high-dimensional parameter spaces

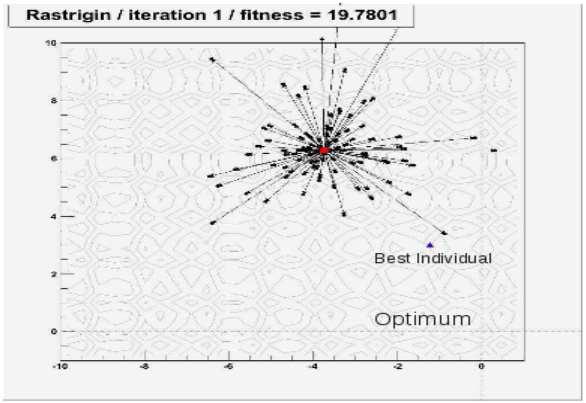
- Parameter scan / DOE only possible for low dimensions of the parameter space
  - Parameter scan with n evaluations per parameter in m dimensions: Need n^m evaluations
  - For 10 parameters and 10 evaluations each at 30 seconds: would require 9500 years CPU time ...
- Thus: need dedicated optimization algorithms that avoid visiting all of the parameter space
- As optimization algorithms will typically call the solver hundreds or thousands of times, such optimization problems will greatly benefit from parallelization

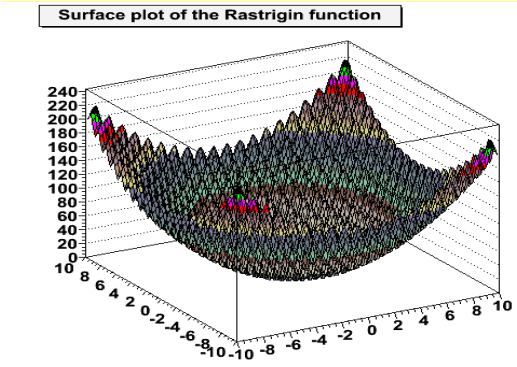


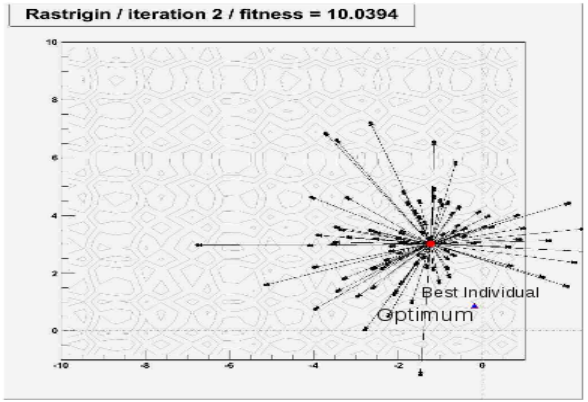
Source: Gemfony

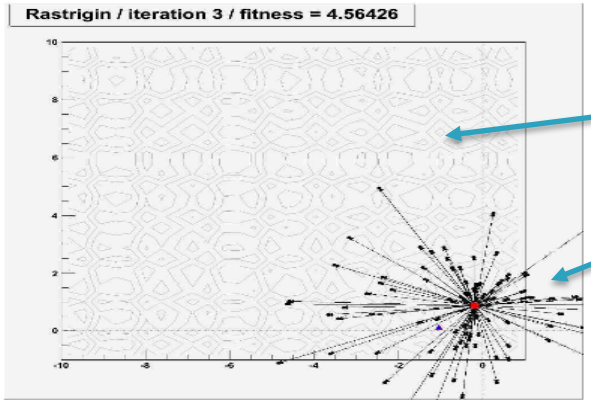
Dealing with Complex Quality Surfaces – Example: Evolutionary Algorithms









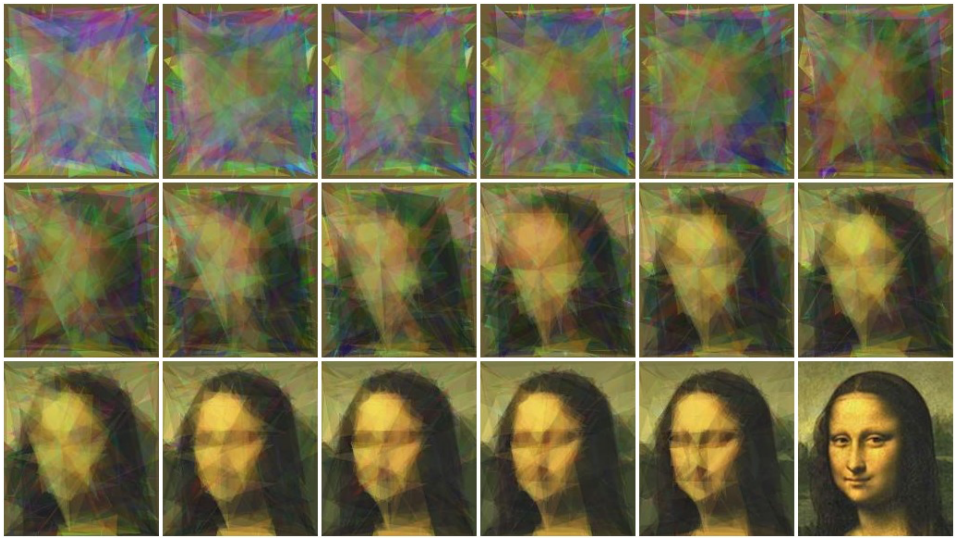


View in –z direction

Parent-individual (red) and children

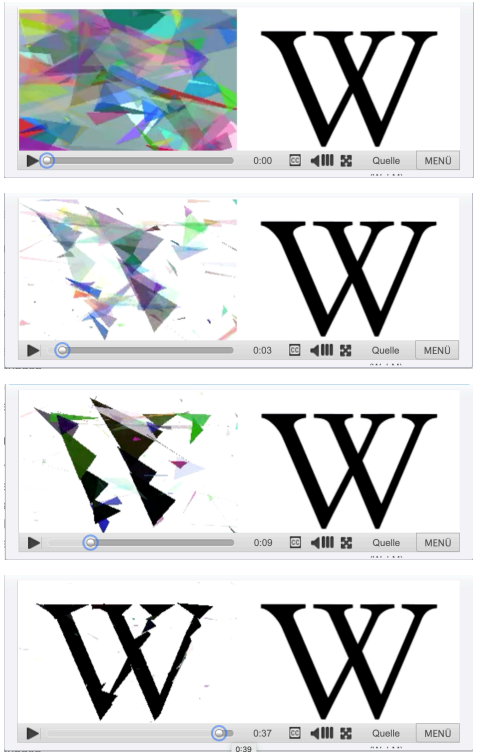
Source: Gemfony

Solving high-dimensional problems with EA



Source: Gemfony

3000 FP Parameters, constrained to [0,1]



Source: https://de.wikipedia.org/wiki/Geneva_(Software)

Asymptotic convergence:
Quick initial successes, followed by slower improvements
3000 parameters marks the limit of usefulness of this EA

Parallelization: Comparatively Easy for Multi-Core Environments

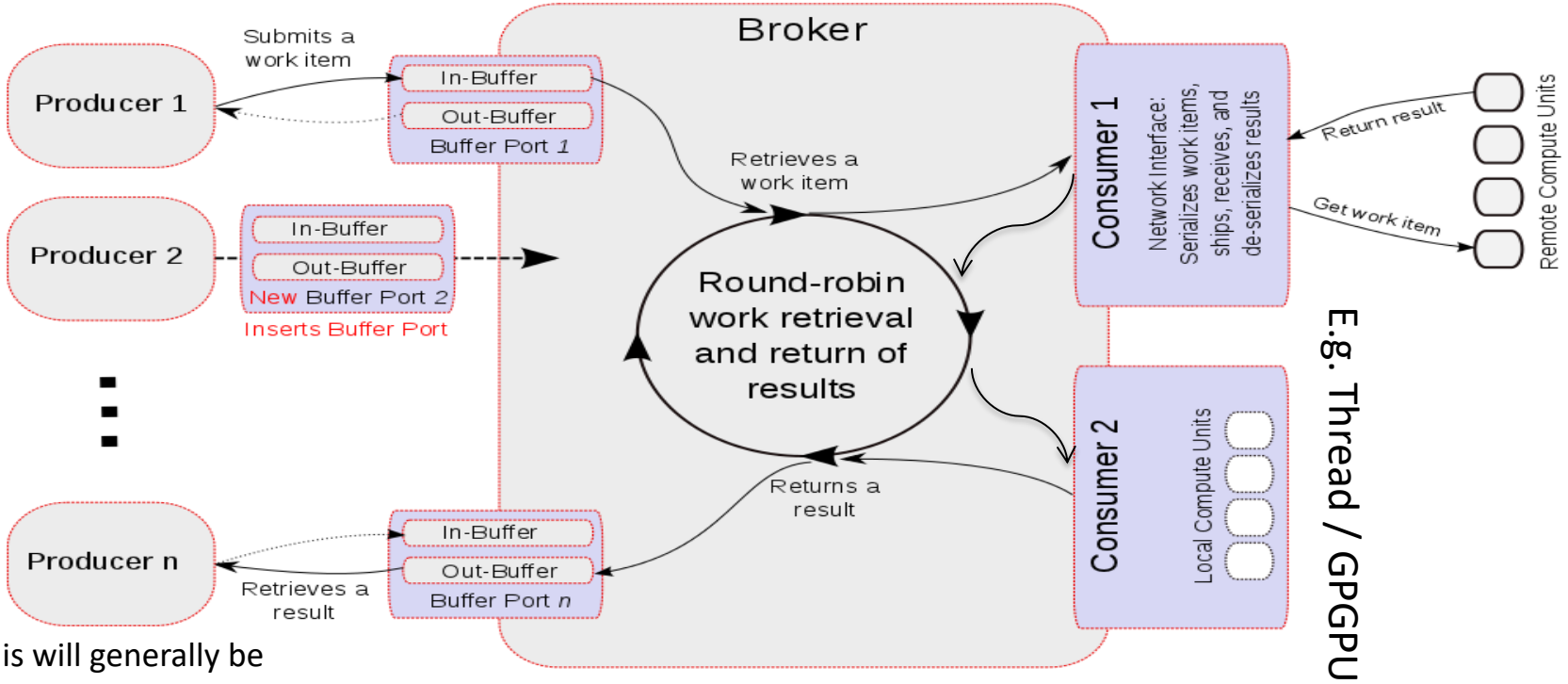
```
Datei  Bearbeiten  Ansicht  Lesezeichen  Einstellungen  Hilfe
top - 16:03:19 up 118 days,  5:21,  2 users,  load average: 44.03, 30.19, 14.96
Tasks: 577 total,   1 running, 576 sleeping,   0 stopped,   0 zombie
Cpu(s): 99.9%us,  0.1%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  98998852k total, 13273640k used, 85725212k free,   366152k buffers
Swap: 101056504k total,    40224k used, 101016280k free, 11412904k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
25879 berlich   17   0 2684m 668m  28m S 4803.0  0.7  97:24.42 GOmegaPiApp
25868 berlich   15   0 13140 1492  820 R    5.8  0.0   0:02.31 top
25603 berlich   15   0 90128 1824 1088 S    0.0  0.0   0:00.08 sshd
25604 berlich   15   0 66060 1620 1252 S    0.0  0.0   0:00.05 bash
25653 berlich   15   0 90128 1828 1088 S    0.0  0.0   0:00.06 sshd
25654 berlich   15   0 66060 1596 1240 S    0.0  0.0   0:00.01 bash
```

- Even large systems may be saturated by suitable work-loads (long evaluation)
- Example uses Evolutionary Strategies, as implemented in Geneva
- For large populations, resembles an “embarrassingly parallel“ problem

Source: Gemfony

Parallelization: More Complex in Distributed Environments

Source: Gemfony



This will generally be
An optimization algorithm

Calculating the timeout in Grid-/Cloud-Environments is complex

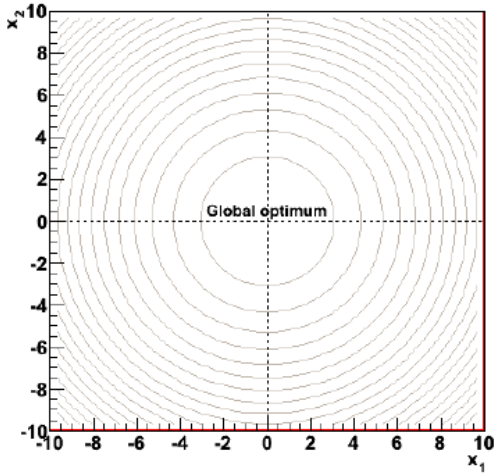
Table of Contents

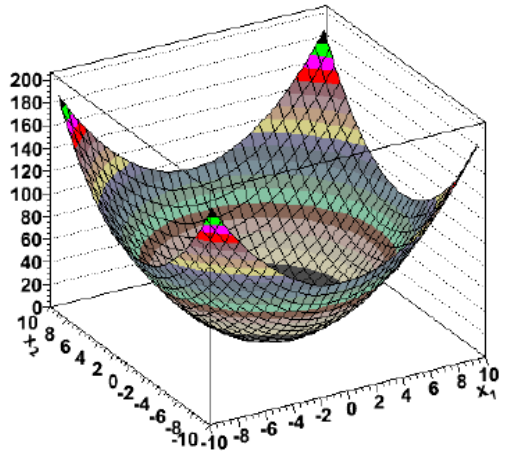
- Introduction and functionality
- Using Geneva
- Geneva application in Hadron theory
- GSI contributions and benchmarking
- Lessons Learned
- Conclusion and outlook

Manual see https://www.gemfony.eu/fileadmin/documentation/geneva-manual.pdf

- Defining a first optimisation problem
- In an n-dimensional paraboloid, the „quality“ of the parameter set (n floating point numbers in this case) is defined as follows:

$$f(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} x_i^2 = x_1^2 + x_2^2 + \ldots + x_n^2$$





Class definition of a 2 dimensional parabola

- The following class lets us search for the minimum of a two-dimensional parabola
- It is derived from GParameterSet, the base class of all individuals

Class GParaboloidIndividual2D

```
class GParaboloidIndividual2D :public GParameterSet
{
public:
  GParaboloidIndividual2D(); // default constructor
  GParaboloidIndividual2D(const GParaboloidIndividual2D&); // copy constructor
  virtual ~GParaboloidIndividual2D(); // destructor

protected:
  // Loads the data of another GParaboloidIndividual2D
  virtual void load_(const GObject*);
  // Creates a deep clone of this object
  virtual GObject* clone_() const;

  // Calculates the object's quality
  virtual double fitnessCalculation();

private:
  // Make the class accessible to Boost.Serialization
  friend class boost::serialization::access;

  // Triggers serialization of this class and its base classes.
  template<typename Archive>
  void serialize(Archive & ar, const unsigned int) {
    using boost::serialization::make_nvp;
    // Serialize the base class
    ar & BOOST_SERIALIZATION_BASE_OBJECT_NVP(GParameterSet);
    // Add other variables here like this:
    // ar & BOOST_SERIALIZATION_NVP(sampleVariable);
  }

  const double PAR_MIN_; // Lower boundary for parameters
  const double PAR_MAX_; // Upper boundary for parameters
};
```

Compare examples in the Geneva distribution (see Github link at the end)

The constructor – adding parameters

Listing 11.2: The constructor of the GParaboloidIndividual2D class

```
GParaboloidIndividual2D::GParaboloidIndividual2D()
        : GParameterSet()
        , PAR_MIN_(-10.)
        , PAR_MAX_(10.)
{
        for(std::size_t npar=0; npar<2; npar++) {
                // GConstrainedDoubleObject is constrained to [PAR_MIN_:PAR_MAX_[
                boost::shared_ptr<GConstrainedDoubleObject>
                   gcdo_ptr(new GConstrainedDoubleObject(PAR_MIN_, PAR_MAX_));
                // Add the parameters to this individual
                this->push_back(gcdo_ptr);
        }
}
```

The fitness calculation

```
double GParaboloidIndividual2D::fitnessCalculation(){
  double result = 0.; // Will hold the result
  std::vector<double> parVec; // Will hold the parameters

  this->streamline(parVec); // Retrieve the parameters

  // Do the actual calculation
  for(std::size_t i=0; i<parVec.size(); i++) {
          result += parVec[i]*parVec[i];
  }

  return result;
}
```

The main function

```
using namespace Gem::Geneva;
int main(int argc, char **argv) {
	Go2 go(argc, argv, "config/go2.json");

	//-------------------------------------------------------------------------
	// Initialize a client, if requested
	if(go.clientMode()) return go.clientRun();

	//-------------------------------------------------------------------------
	// Add individuals and algorithms and perform the actual optimization cycle

	// Make an individual known to the optimizer
	boost::shared_ptr<GParaboloidIndividual2D> p(new GParaboloidIndividual2D());
	go.push_back(p);

	// You could add an algorithm to the Go2 class here, which would always be
	// executed first. Not specifiying any algorithms results in the default
	// default algorithm, unless other algorithms specified on the command line.
```

Using JSON for the configuration Here: GEvolutionaryAlgorithm.json

```
    }
  },
  "maxIteration": {
      "comment": "The maximum allowed number of iterations",
      "default": "1000",
      "value": "1000"
  },
  "minIteration": {
    "comment": "The minimum allowed number of iterations",
    "default": "0",
    "value": "0"
  },
  "maxStallIteration": {
      "comment": "The maximum allowed number of iterations without improvement",
      "comment": "0 means: no constraint.",
      "default": "20",
      "value": "20"
  },
  "indivdualUpdateStallCounterThreshold": {
      "comment": "The number of iterations without improvement after which",
      "comment": "individuals are asked to update their internal data structures",
      "comment": "through the actOnStalls() function. A value of 0 disables this check",
      "default": "0",
      "value": "0"
  },
  "reportIteration": {
      "comment": "The number of iterations after which a report should be issued",
      "default": "1",
      "value": "1"
  },
```

With auto-generated- and user-definable-command line options

```
rberlich@Ubuntu-1910-eoan-64-minimal:~/Progs/geneva-build/examples/geneva/02_GParaboloid2D$ ./GParaboloid2D --help
Usage: ./GParaboloid2D [options]:

Basic options:
  -h [ --help ]                         Emit help message
  --showAll                             Show all available options
  -a [ --optimizationAlgorithms ] arg   A comma-separated list of optimization
                                        algorithms, e.g. "arg1,arg2". 5
                                        algorithms have been registered:
                                        ea:  Evolutionary Algorithm
                                        gd:  Gradient Descent
                                        ps:  Parameter Scan
                                        sa:  Simulated Annealing
                                        swarm:  Swarm Algorithm

  -f [ --cp_file ] arg (=empty)         A file (including its path) holding a
                                        checkpoint for a given optimization
                                        algorithm
  --client                              Indicates that this program should run
                                        as a client or in server mode. Note that
                                        this setting will trigger an error
                                        unless called in conjunction with a
                                        consumer capable of dealing with clients
  --maxClientDuration arg (=00:00:00)   The maximum runtime for a client in the
                                        form "hh:mm:ss". Note that a client may
                                        run longer as this time-frame if its
                                        work load still runs. The default value
                                        "00:00:00" means: "no time limit"
  -c [ --consumer ] arg (=stc)          The name of a consumer for brokered
                                        execution (an error will be flagged if
                                        called with any other execution mode
                                        than (2) ). 4 consumers have been
                                        registered:
                                        asio:  GAsioConsumerT
                                        beast:  GWebsocketConsumerT
                                        sc:  GSerialConsumerT
```

First output

```
Seeding has started
Starting an optimization run with algorithm "Evolutionary Algorithm"
0: 64.6073443050163
1: 25.9597623490252
2: 8.89715425355864
3: 1.45564799125829
4: 0.861887897798893
[...]
999: 7.37074272148514e-13
End of optimization reached in algorithm "Evolutionary Algorithm"
Done ...
```

In the Client Server mode many clients/individuals can run in parallel and contribute to solving a complex problem

Table of Contents

- Introduction and functionality
- Using Geneva
- Geneva application in Hadron theory
- GSI contributions and benchmarking
- Lessons Learned
- Conclusion and outlook

Hadronic reaction amplitudes for FAIR

- A framework for predicting and analysing final-state interactions for the FAIR experiments is beeing developed
- This requires massive parallel computing, up to 50 and more coupled-channels needed
- Reaction amplitudes are derived from effective Lagrangians where coupled-channel unitarity and the implications of micro-causality (dispersion relations) are implemented (isobar models are not good enough)
- Parameter space is reduced significantly by using constraints from chiral and heavy-quark symmetry but also large-Nc QCD
- A subset of the parameters can be derived from the quark-mass dependence of existing QCD lattice data and/or fits to existing data
- Conventional fitting routines like Minuit are not suitable for such problems – gradients are expensive and not stable
- In order to avoid local minima and to be able to find the best possible solution an Evolutionary Algorithm with reasonably high population is under investigation

Projects done with Geneva

Constraints from a large-Nc analysis on meson-baryon interactions at chiral order Q3
Y. Heo, C. Kobdaj, M.F.M. Lutz
Published in: Phys. Rev. D 100 (2019) 9, 094035

On a first order transition in QCD with up, down, and strange quarks
Xiao-Yu Guo, Y. Heo, M.F.M. Lutz
Published in: Eur. Phys. J. C 80 (2020) 3, 260

A generalised Higgs potential with two degenerate minima for a dark QCD matter scenario
M.F.M. Lutz, Y. Heo, Xiao-Yu Guo
Published in: Eur. Phys. J. C 80 (2020) 4, 322

From Hadrons at Unphysical Quark Masses to Coupled-Channel Reaction Dynamics in the
Laboratory
M.F.M. Lutz, Xiao-Yu Guo, Y. Heo
Published in: JPS Conf. Proc. 26 (2019) 022022

Low-energy constants from charmed baryons on QCD lattices
Y. Heo, Xiao-Yu Guo, M.F.M. Lutz
Published in: Phys. Rev. D 101 (2020) 5, 054506

Geneva Cluster @ GSI

example case: 10 minutes compute time for one solution, 10 x 400 clients, 10 x 4000 population, 1000 iterations, one week of compute time in total

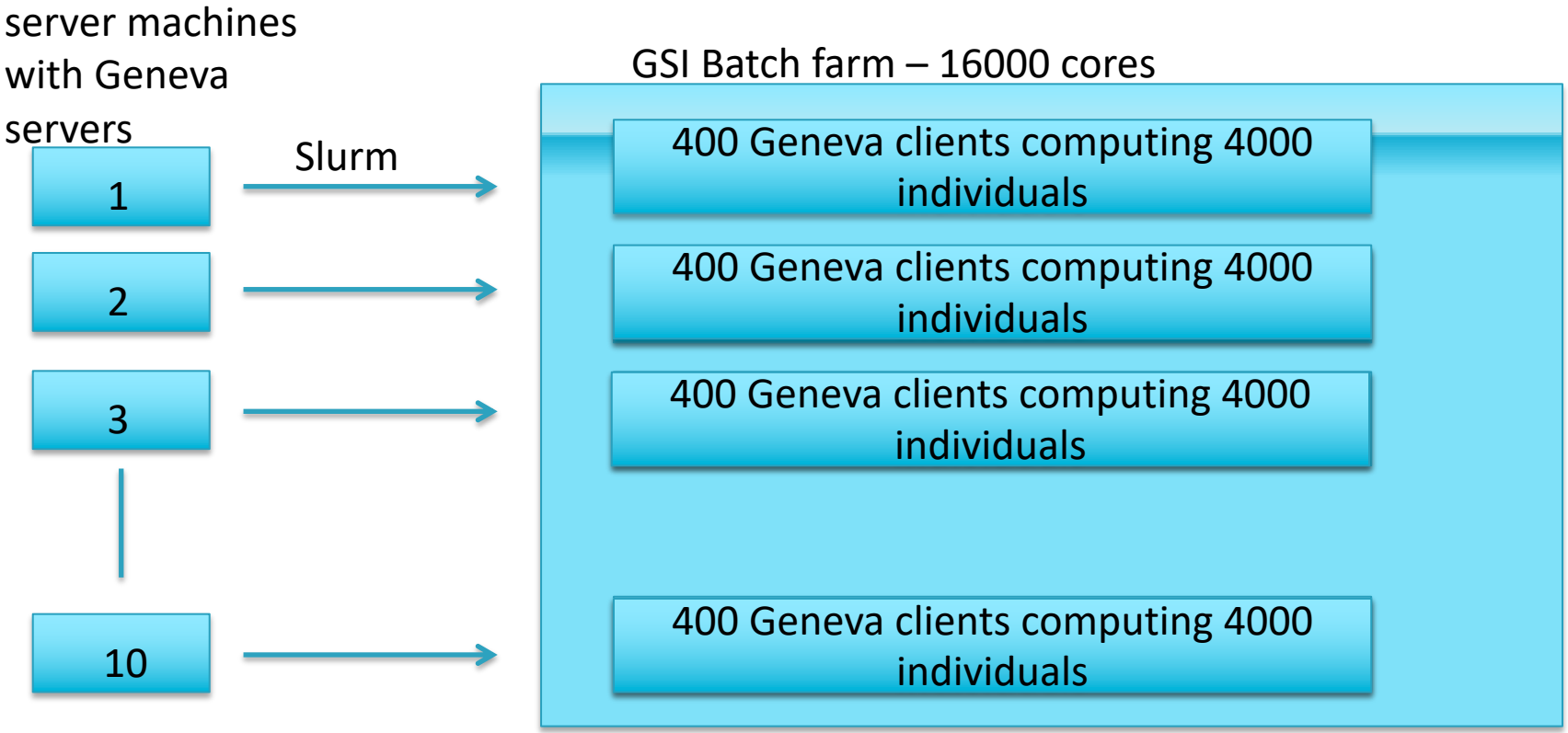


Table of Contents

- Introduction and functionality
- Using Geneva
- Geneva application in Hadron theory
- GSI contributions and benchmarking
- Lessons Learned
- Conclusion and outlook

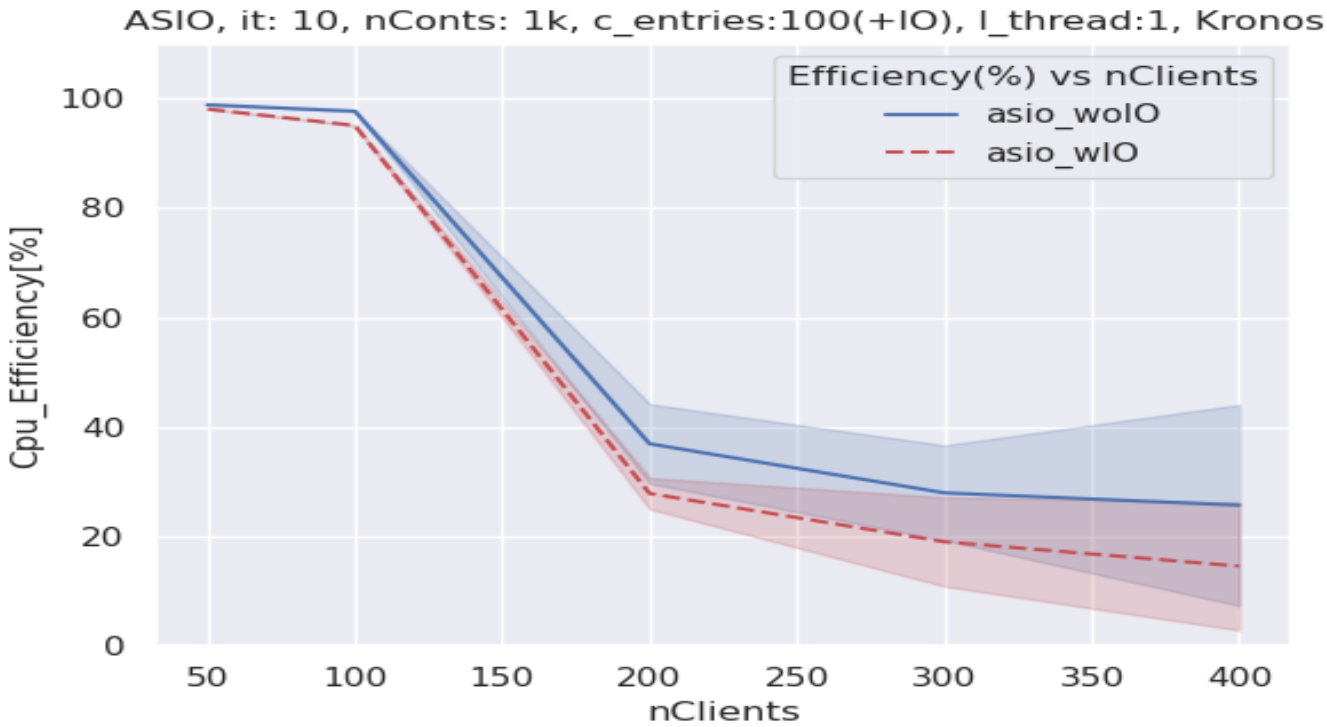
- Geneva client-server communication
  - transition from Boost.ASIO-Sockets to Beast Websockets
  - heatbeat option allows better client control
  - less load on server, higher number of clients, higher efficiency
- Checkpoints
  - iterations are stored in checkpoints (text, xml or binary)
  - iterations can be continued later by loading the checkpoint file

Benchmarks CPU Efficiency vs. nClients

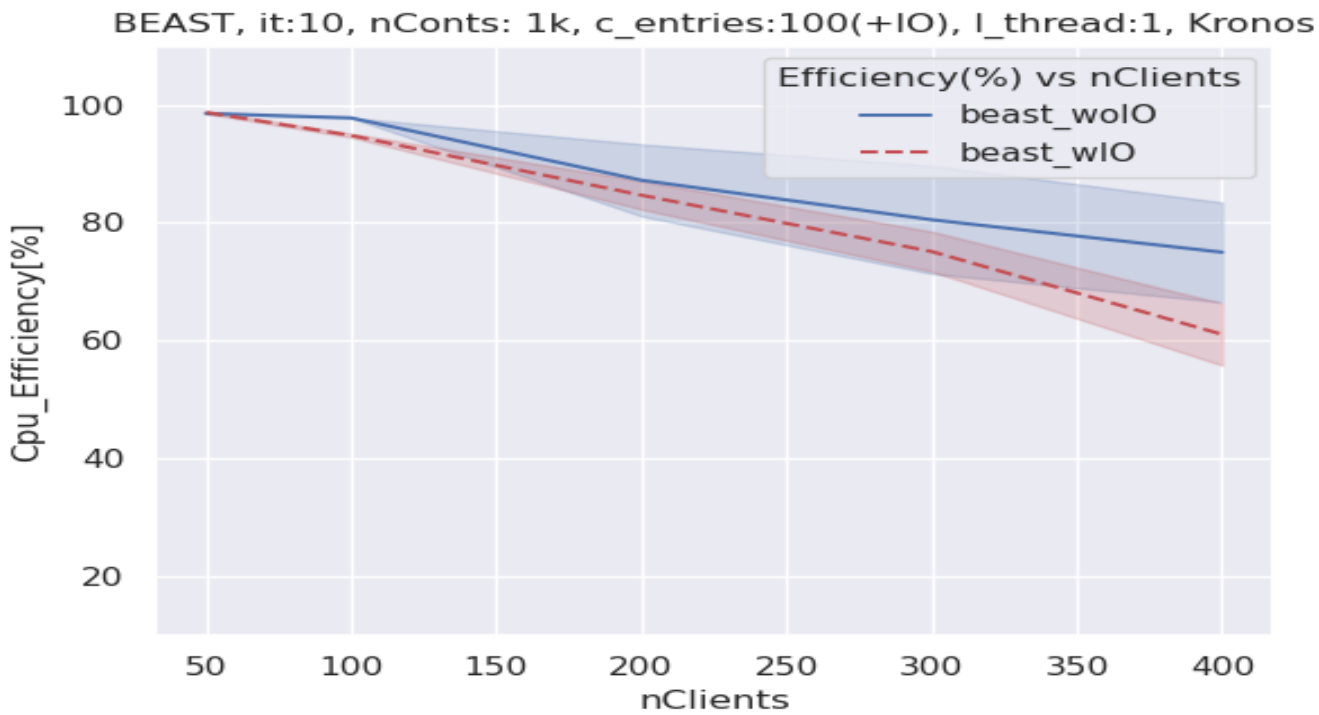
- ongoing developments based on Geneva benchmark suite
- adjusted to Geneva GSI version (May 2019)
  - based on Boost v1.72
- added values
  - Beast WebSocket Consumer/Client added
  - Container classes (Simple, Random) with specific work load (in function process())
  - useful to test changes in the Geneva code base
- automatic script for starting a job in the Cluster

```
dbertini@lxbk0198:/lustre/rz/dbertini/ea/perfs$ ./tracejob.rb -h
Usage: tracejob [options]
    -p, --producers PRODUCERS        number of producers (default 1)
    -w, --workers WORKERS            number of workers (default 1)
    -c, --prod_cycles PROD_CYCLES    number of production cycles (default 2)
    -o, --cont_objs CONT_OBJS        number of container objects (default 10)
    -e, --cont_entries CONT_ENTRIES  number of container entries (default 10)
    -t, --l_threads LISTTHREADS      number of listener threads (default 1)
    -x, --ipc IPC                    networking strategie (default 9)
    -s, --nclients NCLIENTS          number of clients jobs (default 2)
    -q, --queue QUEUE                cluster queue (default main)
    -g, --io IOTEST                  Container IO (default 0)
    -h, --help                       Display this screen
```

ASIO CPU Efficiency and number of clients

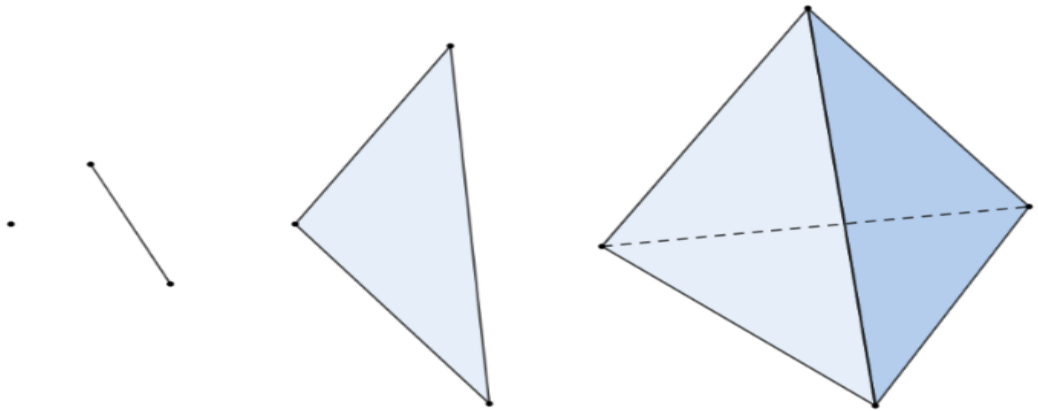


BEAST CPU Efficiency and number of clients

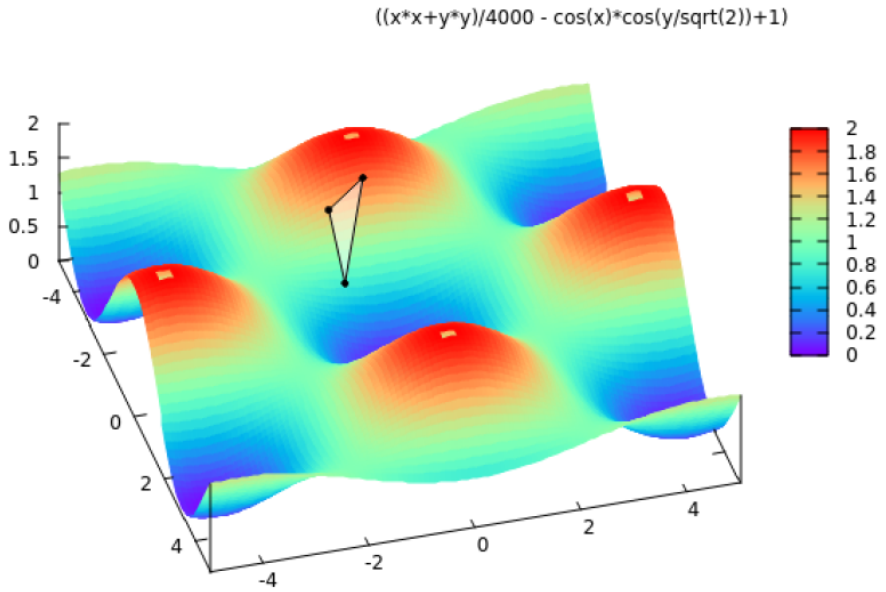


Improvements through GSI

- New: Implementation of a parallel Nelder-Mead-Simplex algorithm
- Based on the geometrical form of the Simplex
- Planned to be included in Geneva



Nelder-Mead-Simplex Algorithm

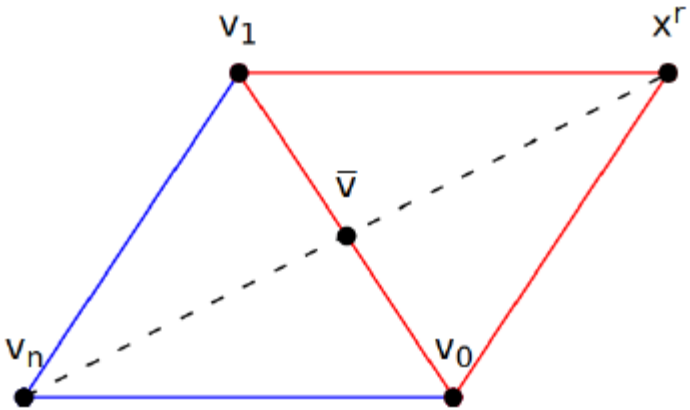


Improvements through GSI

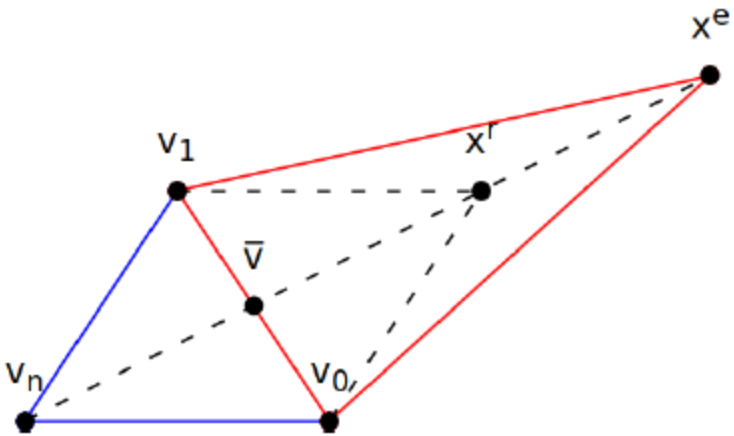
Nelder-Mead-Simplex Algorithm

via the following types of action the geometry of the Simplex is moving through the geometry towards the optimum

1) reflection



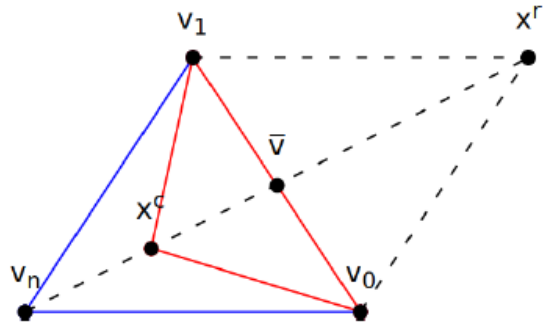
2) expansion

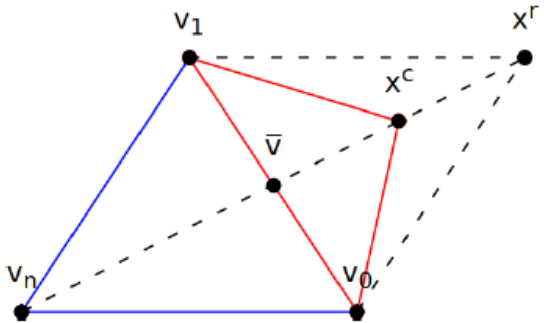


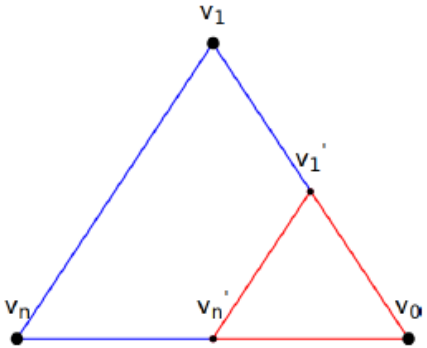
Improvements through GSI

Nelder-Mead-Simplex Algorithm

3) contraction (inwards or outwards)







4) shrinking

Improvements through GSI

Nelder-Mead-Simplex Algorithm

- the parallel simplex algorithm is improving n corners per iteration
- implemented via distributed memory and MPI

Table of Contents

- Introduction and functionality
- Using Geneva
- Geneva application in Hadron theory
- GSI contributions and benchmarking
- Lessons Learned
- Conclusion and outlook

Lessons learned

Geneva has proven to be **highly useful** in many contexts.

But keep in mind the **rough corners**

C++ as an implementation language for distributed systems

- Note: we are not talking about „embarassingly parallel“ here ...
- C++ wants to be the ideal language
  - The “academically perfect“ language
  - High-speed, close to the bare metal
  - All possibilities reserved (but easy to shoot yourself in the foot)
  - „Design by committees“ (plural ...)
- Minimal focus on standard libraries, infrastructure, surrounding
  - No networking in the standard after 40 years ?!??
  - Many missing multithreading constructs (think „threadpool“)
- Language would have long been dead, except
  - There is a large pool of high quality libraries out there (think „Boost“)
  - Highly knowledgable community
  - If you do have the tools and the knowledge, using C++ can be a joy! ☺
- C++ is hence still in wide use throughout science and industry

C++ as an implementation language for distributed systems

- Lack of networking constructs has meant for Geneva: build your own
  - Initially based on MPI, then Boost.ASIO, then Boost.Beast (Websockets)
- Majority of work went into this
  - This was NOT the intention!
  - Not the core business of Geneva (but arguably what makes it useful)
  - Many hard to track bugs, and difficult to find suitable test environments (We are paying > 1000 Euros per year for root servers)
- A useful structure has evolved from modularizing code
  - A random number factory, transparent to the user.
    - Produce centrally when the system is idle or no unused numbers are left in the store.
    - Consume in many parallel entities
  - Brokerage
  - Test-infrastructure (in-class definition, decentral execution)
- Still: could be done (much) better
  - But lets not try to do the same mistake C++ did ...
  - The focus in the future should be much more on optimization algorithms

Licensing

- Make it open
- Make it open
- Make it open
- And use a non-copyleft Open Source license

If I were to do it again ...

- Keep it simple
  - No over-engineering
  - There is a threshold where development efficiency goes down
  - Need modularized code, where each module is easily maintainable by 2 people
- If you want contributions, make it as open as possible.
  - Either it is Open Source or it is not
  - Encourage contributions
- Use seperate languages for specific tasks
  - Do NOT try to make it a monolithic system
  - Implies decoupled duties and tasks
  - Idea: MQTT broker, allow algorithms to be developed in any language
- Concentrate on the core tasks, do not try to improve languages
- Test first, then develop
- Refactoring
  - Reuse your own, good ideas (but re-arrange and re-write them)

This can not be done alone ...

Table of Contents

- Introduction and functionality
- Using Geneva
- Geneva application in Hadron theory
- GSI contributions and benchmarking
- Lessons Learned
- Conclusion and outlook

Conclusion

- Geneva is an efficient client/server tool for doing distributed optimisation within an HPC environment
  - mainly using Evolutionary Algorithm
  - using up to 400 clients with population up to 4000
- May need refactoring and needs a larger community
  - Needs ideas both for optimization and for the clustering part
- Future work
  - adding more reliable optimisation algorithms
  - increasing scalability
  - starting inter-site optimisation on Grids/Clouds
  - Geneva Spack package and Geneva Singularity Container for easy use
  - Simplify Geneva interface even more for common usage patterns
- We are highly interested in pointers and contributions regarding the extension of Geneva towards ML

Thank you

Do contact us in case of questions:

k.schwarz@gsi.de
r.berlich@gemfony.eu

If you want to try Geneva:

https://github.com/gemfony/geneva

Masthead Gemfony

Address	Gemfony scientific UG (haftungsbeschränkt) Hauptstraße 2 76344 Eggenstein-Leopoldshafen - Germany
Telefone	+49(0)7247/934278-0
Fax	+49 (0)7247 934 2781
Email	contact@gemfony.eu
Registered at	Amtsgericht Mannheim (Germany)
Registration-Id	HRB 710566
Ust.-Id	DE274421406
Managing Director	Dr. Rüdiger Berlich

Material used

Slide	Figure	Source
The Geneva library collection	Car in puzzle	Image courtesy of Simon Howden at FreeDigitalPhotos.net
The Geneva library collection	Wind turbines in puzzle	http://www.flickr.com/photos/pebondestad/3533177131/sizes/l/in/photostream/Creative Commons Attribution 2.0; By Pål Espen Bondestad.
The Geneva library collection	Particle decay	https://en.wikipedia.org/wiki/File:CMS_Higgs-event.jpg ; Creative Commons Attribution Share-Alike 3.0; By CERN
Other slides	Other pictures	Gemfony scientific + GSI, or marked on the page